PageRank on an evolving graph

Article · August 2012

DOI: 10.1145/2339530.2339539

CITATIONS
57

READS
935

4 authors:

Bahman Bahmani
Shahid Rajaee University
14 PUBLICATIONS **1,326** CITATIONS
SEE PROFILE

Ravi Kumar
Independent Researcher
256 PUBLICATIONS **25,468** CITATIONS
SEE PROFILE

Mohammad Mahdian
Google Inc.
56 PUBLICATIONS **4,197** CITATIONS
SEE PROFILE

Eli Upfal
Brown University
272 PUBLICATIONS **14,542** CITATIONS
SEE PROFILE

PageRank on an Evolving Graph

Bahman Bahmani*
Computer Science
Stanford University
Stanford, CA
bahman@stanford.edu

Ravi Kumar
Yahoo! Research
701 First Avenue
Sunnyvale, CA
ravikumar@yahoo-inc.com

Mohammad Mahdian*
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA
mahdian@gmail.com

Eli Upfal†
Computer Science
Brown University
Providence, RI
eli@cs.brown.edu

ABSTRACT

One of the most important features of the Web graph and social networks is that they are constantly evolving. The classical computational paradigm, which assumes a fixed data set as an input to an algorithm that terminates, is inadequate for such settings. In this paper we study the problem of computing PageRank on an evolving graph. We propose an algorithm that, at any moment in the time and by crawling a small portion of the graph, provides an estimate of the PageRank that is close to the true PageRank of the graph at that moment. We will also evaluate our algorithm experimentally on real data sets and on randomly generated inputs. Under a stylized model of graph evolution, we show that our algorithm achieves a provable performance guarantee that is significantly better than the naive algorithm that crawls the nodes in a round-robin fashion.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications – Data Mining

Keywords: Dynamic graphs, PageRank, Random walks

1. INTRODUCTION

The traditional paradigm in the field of algorithm design assumes that the algorithm reads the input data set, produces a solution for this input, and then terminates. Unfortunately, this simplistic model is inadequate for many of today's data mining applications, particularly applications on the Web and in social networks. Here, the input data is not stationary. Rather, it changes over time and the algorithm, unaware of these changes, needs to probe the input continually and produce solutions at any point in time that are close to the correct solution for the then-current input. A formalization of this framework was introduced by Anagnostopoulos et al., who obtained efficient algorithms for sorting [1] and several classical graph problems [2].

In this work we focus on a problem that is probably the best fit to this framework: computing PageRank on an evolving graph. Since its groundbreaking introduction [23], PageRank has emerged as a very effective measure of reputation for both web graphs and social networks. Despite many advances in the field of web information retrieval, search engines still rely on variants of the notion of PageRank for ranking search results. PageRank is a quantity that is computed based on the graph structure of the web. This graph structure changes over time, and the only way for the search engine to learn about the new changes is to crawl the web, i.e., look up a web page (a node in the graph) to discover all its outgoing links. There is a limit on the crawling capacity, and therefore the search engine often has a stale image of the graph. The objective is to design an algorithm that decides which pages to crawl and computes the PageRank using the obtained information in such a way that the error in the computed PageRank (as compared to the PageRank of the current graph) is bounded at any point in time. PageRank-like measures have applications far beyond search, for example, in computing trust and reputation in online social networks; see [6] for more details.

Several algorithms for updating PageRank values upon changes to the network have been proposed; see Section 2. However, all of these algorithms assume knowledge of *all* the changes in the network. But, this assumption is unrealistic in many modern applications of PageRank. For instance, a web search engine using PageRank in its search rankings can learn about the changes to the web graph only through periodically crawling the web. Similarly, a third-party vendor computing PageRank on a constantly evolving social network such as Twitter or Facebook can only learn about the changes to the network via a rate-limited API.

Our contributions. In this paper we study the problem of computing PageRank in the evolving graph framework. The graph slowly changes over time and the algorithm can

*Part of this work was done while the author was at Yahoo! Research.

†Part of this work was done while the author was visiting Yahoo! Research.

KDD'12, August 12–16, 2012, Beijing, China.

observe the graph by periodically probing its nodes to obtain its current set of outgoing neighbors. The goal is to design a probing algorithm that maintains a good approximation of the true PageRank values of the underlying evolving graph.

We propose two algorithms, one randomized and one deterministic, for solving this problem (Section 4). The randomized algorithm is based on the intuition that nodes with higher PageRank values need to be probed more often since they have more impact on the PageRank values of other nodes; the deterministic algorithm also embodies this intuition, but with additional guarantees. We then evaluate our algorithms empirically on real and randomly generated datasets. We demonstrate that they also perform well in practice when compared to two natural baselines of random probing and round-robin probing (Section 5).

Under a stylized yet arguably natural model of graph evolution, we give a theoretical analysis of the performance of the randomized algorithm and prove that it significantly outperforms the simple algorithm that crawls all pages with equal frequency (Section 6). Finally, we sketch how our theoretical results can be adapted and generalized to variants and extensions of our simple model (Section 6.3).

2. RELATED WORK

Efficient computation of PageRank and its variants on massive graphs has attracted a great amount of interest over time in both academic and industrial settings, and algorithms in different computational models have been proposed; see the survey on PageRank [6]. Broadly speaking, algorithms for PageRank fall into one of two categories: linear algebraic methods [7, 28, 13, 14, 15, 8, 11, 17, 18, 19] and Monte Carlo methods [3, 9, 5, 4]. The simplest linear algebraic method is the basic Power Iteration [23] and typical algorithms in this category aim to speed up the power iteration through a combination of linear algebraic techniques and careful use of the properties of the practical graphs, e.g., the web graph [13, 14, 15]. Monte Carlo methods have the advantage of being very efficient and highly scalable [3]. Hence, they have been adapted to design algorithms for scalable computation of PageRank and its variants in emerging computational models such as data streaming [9] and MapReduce [4].

Algorithms for updating PageRank values upon changes to the network have been proposed in both the broad categories mentioned above. A family of the linear algebraic methods use the "aggregation" idea to efficiently update the PageRank vector [8, 17, 18]. Tong et al. [29] and McSherry [20] also propose algorithms to deal with PageRank updates. Among the Monte Carlo methods, Bahmani et al. [5] proposes an incremental algorithm for computation of PageRank, which is fast enough even for real-time updates. However, all of these algorithms, which belong to either the data streaming [21] or dynamic graph [12] computation models, assume knowledge of all the changes in the network. Our algorithm, on the other hand, must work without being notified of the underlying changes to the network.

The evolving model of computation was introduced and formalized by Anagnostopoulos et al. [1, 2]. In contrast with more traditional computation models such as streaming [21], property testing [26, 27], multi-armed-bandits [16], and dynamic graph models [12], this model captures both of the two essential properties of large evolving data applications: the input is changing over time without notifying the algorithm, and the only way for the algorithm to learn about these changes (to accordingly update its solution) is by periodically probing the input in a limited way. Anagnostopoulos et al. [1, 2] formalize this framework and study sorting, selection, and basic graph connectivity problems in this computation model.

Note that there is a line of research on algorithms for computing optimal crawling schedules on the web [30, 24, 10, 22, 25]. The objective in these papers is generally to put together a crawling schedule that minimizes the staleness of the cached image of the web. This is different from our objective, which is to compute the PageRank with as little error as possible.

3. BACKGROUND

Evolving graph model. We first define the model of computation on evolving graphs that will be used throughout this paper. Our model follows the general framework proposed in [1, 2]. We deliberately do not specify how the graph changes in our model in order to keep the model itself general and to emphasize that our algorithms apply to this level of generality. Later, in Section 6, we will fix a graph change model for the purposes of theoretical analysis.

We assume a discrete notion of time, indexed by non-negative integers. The directed graph at a point t in time is denoted by G^t. We assume that an upper bound on the total number of nodes of the graph is known ahead of time and is a fixed quantity n. We stress that this assumption is only for the simplicity of analysis and in Section 6.3 we will discuss how this assumption can be relaxed; our algorithms actually do not need this assumption. The sequence $\{G^t \mid t \geq 0\}$ thus defines a graph that changes over time. We assume that the difference between two consecutive graphs in this sequence is small; this assumption is without loss of generality since it can be achieved by choosing a sufficiently fine time granularity.

An algorithm in this model can observe the graph by *probing* its nodes. At each time step the algorithm can probe at most one node,* upon which it will see the current set of outgoing edges of this node. Since the graph is changing and the algorithm can only see a small part of the graph through each probe, it cannot have a completely up-to-date image of the graph. Note that in our model we do not impose constraints on the running time or the storage space of our algorithm. Such constraints, even though interesting and meaningful, lead to completely different aspects of the problem that are well-explored in the literature.† Therefore, the focus of our model is on the strategy for deciding which node to probe each time.

PageRank. We now recall the definition of PageRank. Let ϵ be a fixed constant called the *teleport probability* (probability of jumping to a random node). The PageRank of a graph G is a vector π that is the stationary distribution of the following random walk: at any step of the walk, with probability ϵ we move to a node chosen uniformly at random

*This is once again for simplicity of exposition. It will be easy to extend the algorithms and the model to the more general case.

†In fact, since the main time-consuming component of our algorithms is the computation of PageRank, they can be made efficient using the previous work on efficient PageRank computation.

and with probability $1-\epsilon$, we choose one of the outgoing edges of the current node uniformly at random and move to the head of that edge.[‡] The vector π is called the *PageRank* vector of G, and π_v is called the PageRank of node v in G. Denote by π_u^t, in_u^t, and out_u^t the PageRank, the in-degree, and the out-degree of node u in G^t.

Recall that our task is to design an algorithm to compute the PageRank of a directed graph that is changing. At the end of each time step, the algorithm should output a PageRank vector ϕ^t for the graph. The goal is to do this in such a way that the difference between π^t and ϕ^t is small.

4. ALGORITHMS

In this section we describe the various algorithms for computing PageRank on an evolving graph. Recall that each algorithm essentially boils down to a probing strategy: in what order the nodes in the graph should be probed to learn the neighborhood of the node. Each of our algorithms will operate by maintaining an image of the graph that it learns according to its probes. It computes PageRank on this maintained image and outputs that as its approximation to the true PageRank. Note that this automatically determines the time and space complexities of the algorithm: it needs to store the entire graph as it sees and can use power iteration to compute the PageRank. The primary quantity of interest is therefore the error caused by a particular probing method. We describe two baseline algorithms where the probe strategy is trivial and two algorithms that use a non-trivial probing strategy. In Section 5, we will experimentally evaluate these algorithms.

Baselines. We start by defining two algorithms that act as baselines used for comparison in Section 5. The first algorithm probes a node chosen uniformly at random at every time step. Based on these probes, at each point t, it has a most recent image H^t of the graph (i.e., the set of out-going edges of v in H^t is the set observed the last time v was probed). The output is the PageRank vector of this image. We call this algorithm *Random Probing*.

Random Probing is based on probing nodes with equal frequencies. An alternative, deterministic, way to implement this is by cycling through the nodes and probing them in this order. Again, the output at any moment is the PageRank vector of the current image of the graph. This algorithm is called *Round-Robin Probing*.

Proportional Probing. To improve over the above baselines, we need to depart from probing nodes with equal frequencies. The intuition is that if the outgoing edges of a node of high PageRank changes, then it affects the PageRank of other nodes by a larger amount. Therefore, such nodes should be probed with higher frequency. This motivates our first algorithm, which we call *Proportional Probing*:

> At each step t, pick a node v to probe with probability proportional to the PageRank of v in the algorithm's current image of the graph. The output, as before, is the PageRank vector of the current image.

For the purpose of the theoretical analysis (in Section 6), we analyze a variant of the Proportional Probing that alternates between probing nodes in a round-robin fashion and probing nodes chosen randomly with probabilities proportional to their PageRank values in the current image. This alternation, while does not necessarily help in practice, makes it easier to analyze the accuracy of the algorithm.

Priority Probing. Finally, we give a deterministic algorithm that probes nodes with frequencies proportional to their current PageRank. The idea is to define a *priority* for each node. Initially, the priority of each node is to zero. In each time step, the node with the highest priority is probed (or if there is more than one, one is arbitrarily chosen). The priority of this node is set to zero, and the priority of other nodes is incremented by their PageRank in the current image of the graph. This pseudo-code is presented in Algorithm 1. We call this algorithm *Priority Probing*.

nodes u $\text{priority}_u \leftarrow 0$ every time step t $v \leftarrow \arg\max_u \text{priority}_u$ Probe v Let H^t be the current image of the graph Output the PageRank vector ϕ^t of H^t $\text{priority}_v \leftarrow 0$ nodes $u \neq v$ $\text{priority}_u \leftarrow \text{priority}_u + \phi_u^t$

: Priority Probing

The main advantage of Priority Probing over Proportional Probing is the following. Proportional Probing picks nodes proportional to their PageRank, but in a stochastic manner. Priority Probing arose out of the following question: is there a (deterministic) probing method such that at each step each node u is picked with the prescribed probability ϕ_u^t and is picked at least once every $O(1/\phi_u^t)$ steps? Note that such a probing strategy will ensure that the algorithm will never miss probing a set of nodes for too long because of the randomness.

It is easy to show using the probabilistic method that such a probing sequence always exists. Priority Probing is a just a simple constructive way of realizing such a sequence. To analyze the guarantee of Priority Probing, for simplicity of analysis, assume that all the ϕ_u^t's are different; therefore, if a node u ever reaches $\text{priority}_u = 1$, then it will be probed in the next step. For any $p \in (0, 1]$, there can be no more than $1/p$ nodes with $\phi_u^t \geq p$; this follows since priority_u of each unprobed node u is incremented by ϕ_u^t. Thus, for any node u, by the time priority_u becomes 1, all nodes v such that $\phi_v^t > \phi_u^t$ would have already been probed.

5. EXPERIMENTS

In this section we present the results of our experiments on real-world graphs to study the performance of our algorithms. We start by describing the data set in Section 5.1. Then, in Section 5.2, we will describe the experiments, i.e., the algorithms we run, the parameters we will let vary, and the metrics we will measure. The results are presented in Section 5.3 and the figures therein.

5.1 Data

In this section we describe the data sets used in our experiments. Each data set is comprised of two parts: the initial graph and evolving graph. The initial graph is the graph we use to bootstrap the process. The evolving graph is a sequence of additions and deletions of temporal edges and nodes of the graph. Now, we proceed to describe the data sets. Table 1 shows their statistical characteristics.

AS. The graph of routers comprising the Internet is organized into subgraphs called Autonomous Systems (AS). Each AS exchanges traffic flows with some of its peers and

[‡]If the current node has no out-going edge, we jump to a random node with probability 1.

one can construct a communication network from the BGP (Border Gateway Protocol) logs. This data was collected from the University of Oregon Route Views Project and consists of 733 daily instances from November 8, 1997 to January 2, 2000. The dataset is publicly available and can be downloaded from `http://snap.stanford.edu/data/as.html`. We fix the communication graph on November 8, 1997 to be the initial graph and we assume that each link in the communication graph lasts for a day, i.e., if a directed edge is present on the ith day and not on the $(i+1)$st day, we assume that it was deleted on the $(i+1)$st day. Likewise, a directed edge is assumed to be added on the $(i+1)$st day if it was not present on the ith day.

CAIDA. This dataset contains 122 CAIDA Autonomous Systems graphs from January 2004 to November 2007 (see `http://www.caida.org/data/active/as-relationships/`). This dataset is also publicly available from `http://snap.stanford.edu/data/as-caida.html`. As before, this graph denotes the communication patterns of the routers. We take the initial graph to be the communication in January 2004 and we assume that each link exists until the next snapshot. The interpretation of edge addition/deletion is as in AS.

RAND. This is a data set that is randomly generated according to the distribution described in Section 6. We generated the graph with $n = 100$ nodes and out-degrees that form a power law with exponent 0.5 (i.e., the out-degree of the ith node is proportional to $i^{0.5}$). We simulated the model for 500000 changes, i.e., 250000 edge removals and the same number of edge additions.

Dataset	max #nodes (n)	#initial edges	#temporal edges	%edge additions
AS	7,716	10,696	488,986	0.516
CAIDA	31,379	65,911	1,084,388	0.518
RAND	100	715	250,000	0.5

Table 1: Details of the datasets used.

5.2 Experiments

Our goal is to evaluate the four algorithms described in Section 4. Of these four algorithms, Random Probing and Round-Robin Probing serve as baselines, and our goal is to compare Proportional Probing and Priority Probing against these baselines. (As we mentioned earlier, we know of no incremental PageRank algorithm that works in our model, i.e., when the algorithm is not aware of the changes to the graph.) In a sense, Random Probing serves as a baseline for Proportional Probing (as they are both randomized and can select which node to probe in an efficient and parallelizable way), and Round-Robin serves as a baseline for Priority Probing (as they are both deterministic). One can expect the deterministic algorithms to better than the corresponding randomized algorithms, as they avoid some of the "waste" caused by each time flipping a coin independent of the previous steps.

In addition to these algorithms, we run a *hybrid* between Proportional Probing and Round-Robin Probing. The hybrid algorithm is parametrized by a parameter $\beta \in [0,1]$ which controls the ratio of round-robin vs proportional probes made by the algorithm; making $\beta = 1$ will be the same as Round-Robin Probing. This is essentially a generalized version of the algorithm analyzed theoretically in Section 6.

We assume that initially all the algorithms have an image of the graph that is a perturbed version of the graph at that point. This is done to ensure that the simulations reach a steady state quickly. If we initialize the simulation assuming that the algorithms can observe the initial graph, the error will start at zero and will reach the steady state after some time. Similarly, if we start the algorithms without any knowledge of the graph, the error initially will be high, and will decrease to the steady state over time (although this process would take long). At the end, what we are interested in is the error of the algorithms after they stabilize.

For most of our simulations, we assume that the change process and the probing process progress at the same rate, i.e., for each change the algorithm can probe once. More precisely, we will have a *recomputing frequency* K, which indicates that after each K steps, the algorithm gets to do K probes. This is enough for simulations that seek to compare the algorithms. We will also experiment with varying the relative speed of the change and probing frequencies. This will be parameterized by a number α, which indicates the number of probes allowed for the algorithm for every (edge) change; here, an addition or deletion of an edge is considered to be a change. This means that after every K changes to the graph, the algorithm gets to make αK probes.

Metrics. Our main measure of performance is the L_∞ metric, which is defined as $L_\infty(\pi^t, \phi^t) = \max_{u \in V} |\pi^t(u) - \phi^t(u)|$, where π^t is the PageRank vector of the current graph, and ϕ^t is the vector output by the algorithm (both of these vectors are unit vectors in the ℓ_1 norm). To show that our results are "robust" to the choice of performance metric, we also run some of the simulations with the L_1 metric, which is defined as $L_1(\pi^t, \phi^t) = \sum_{u \in V} |\pi^t(u) - \phi^t(u)|$.

5.3 Results

A comparison of the algorithms. As expected intuitively, in both data sets AS and CAIDA and according to both measures L_1 and L_∞, the performance of Proportional Probing is almost always better than Random Probing (i.e., has lower error), and Priority Probing is almost always better than Round-Robin Probing. There is no clear comparison between Proportional Probing and Round-Robin Probing. The results are presented in Figure 1 (for the AS graph) and Figure 2 (for the CAIDA graph). As can be seen, the difference between the algorithms can be significant; for example, Priority and Random Probing have errors that are almost a factor two apart in the CAIDA data set. For RAND the difference is even greater, but this is to be expected given our theoretical analysis in Section 6. Note that the numbers on the y-axis in all these figures are small since the PageRank vector is normalized; the numbers should be viewed with the magnitude of n (from Table 1) in mind.

We also ran the experiments with the probing strategy proportional to the in-degree of a node instead of its PageRank value. All of these results were qualitatively identical to the PageRank results and we omit them.

Effect of probing rate α. Figure 4 shows the results of the simulations on the AS graph with $\alpha = 2, 4$. Obviously, the algorithms perform better when they probe more frequently. Furthermore, the choice of α does not seem to affect the comparison between different algorithms, i.e., they are often ordered the same way they were for $\alpha = 1$ (Figure 1).

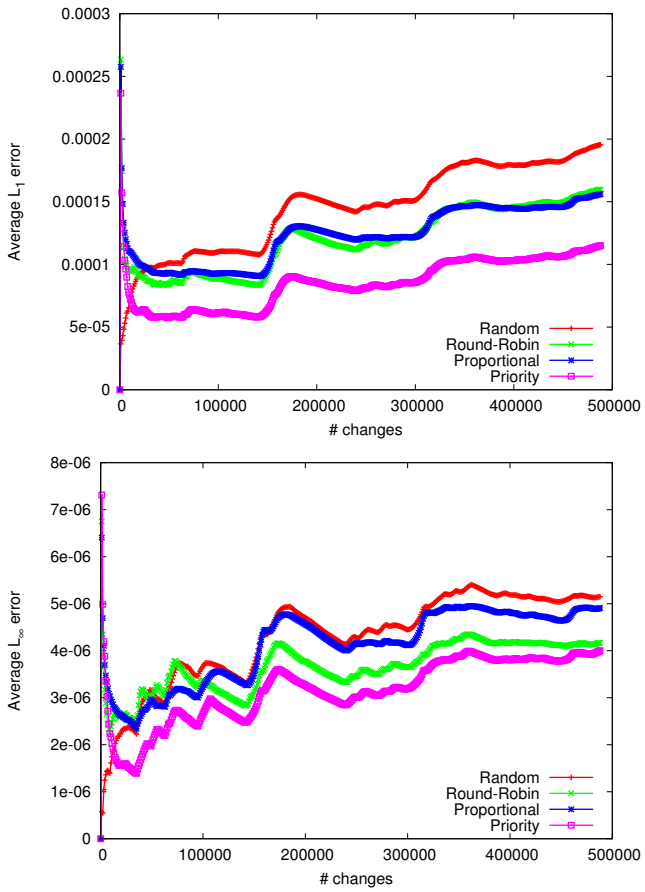

Figure 1: Average L_1 and L_∞ error of various algorithms for the AS graph.

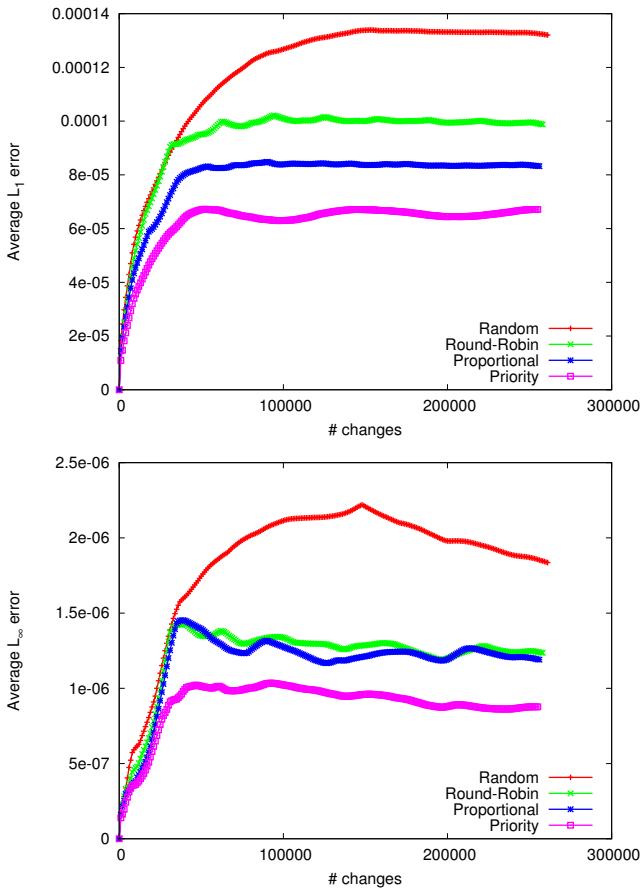

Figure 2: Average L_1 and L_∞ error of various algorithms for the CAIDA graph.

Algorithm's image vs truth. Another curious question is which of the algorithms has a more "stale" image of the graph. This can be measured by counting the number of edges that are in the image but not in the true graph, and vice versa. The result, for the AS graph, is shown in Figure 5. This figure suggests that there is no clear comparison between the staleness of the images of the four algorithms. This is interesting, since if the changes were happening uniformly at random in the graph, then Random Probing and Round-Robin Probing should have had an advantage in terms of the staleness of their image, whereas we see that quite often this is not the case. This suggests that the change in real-world networks is biased toward nodes with higher importance, and this gives a double advantage to Proportional Probing and Priority Probing.

The hybrid algorithm. We run the hybrid algorithm that interpolates between Round-Robin Probing and Proportional Probing with different parameters to see the effect of the round-robin rate β on the error. As can be seen in Figure 6, the lowest error happens at approximately $\beta = 0.9$. It is curious that this hybrid algorithm performs better than both Round-Robin Probing and Proportional Probing. Perhaps one intuitive reason for this is that the hybrid algorithm assigns more frequency to the more important nodes, while at the same time guaranteeing that each node will be visited at least once during the last $O(n)$ steps. This is precisely the property of the hybrid algorithm that is used in the theoretical analysis.

6. A THEORETICAL ANALYSIS

In this section we provide a theoretical analysis of the hybrid algorithm that alternate between proportional and round robin probing. In order to give a meaningful analysis of the error bound, we need to first fix a model for how the graph changes over time. We do that in the following section by defining a natural stochastic graph process.

6.1 Model and notation

There are many ways to define a model for how G^t changes into G^{t+1}. Intuitively, we want a model that is simple, captures the fact that links on the web change, and that this change happens in a non-uniform way, i.e., popular nodes have higher probabilities of being linked to. At the same time, we also want a near-realistic model where we can something formal and rigorous.

As stated before, we assume that the graph does not grow, i.e., the number of nodes of the graph is fixed to be n. Also, we assume that the number of edges does not change in this process. Both these assumptions are for simplicity and we discuss in Section 6.3 how they can be relaxed. At any time instance, we pick one link at random, and change the page it is pointing at to a new page. The new page is picked with

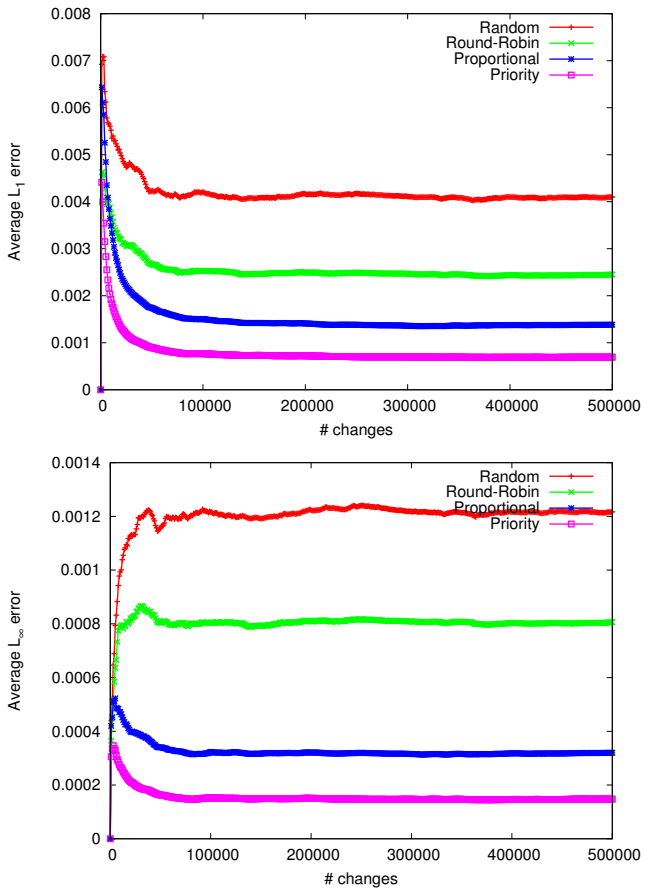

Figure 3: Average L_1 and L_∞ error of various algorithms for the RAND graph.

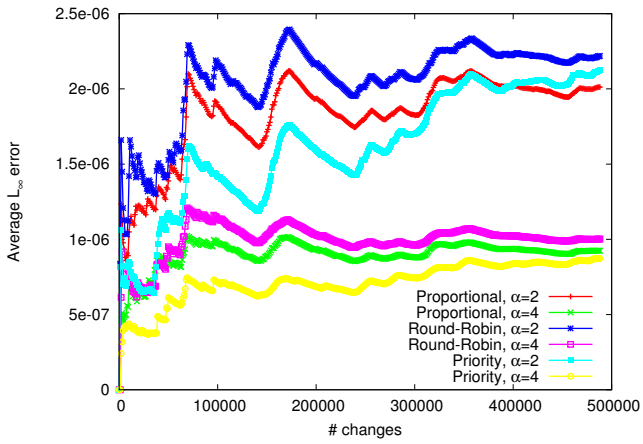

Figure 4: Average L_∞ error of Round-Robin, Proportional, and Priority as a function of the probing rate α for the AS graph.

probability proportional to its current PageRank, which intuitively captures the importance of this page at that time.

Formally, we define a stochastic directed graph process $\{G^t \mid t \geq 0\}$ where G^0 is an arbitrary directed graph on n nodes and m edges and graph G^{t+1} is obtained from G^t by the following process: an edge (u, v) is chosen uniformly at random from the m edges and its head is moved to a new node v' chosen with probability equal to $\pi^t_{v'}$.

Note that the implicit assumption here is that the rate of probe of our algorithm is the same as the rate of change (since each step corresponds to one probe). This assumption is for simplicity and our analysis carries over to the case that the ratio of the rate of the two processes is any constant, with the appropriate constants modifying the bounds.

Let $P(t)$ be the matrix of a random walk on G^t, where $P(t)_{u,v} = 1/\text{out}^t_u$ if there is a direct edge from u to v, otherwise $P(t)_{u,v} = 0$. Le $\bar{1}$ denote the all-ones vector.

6.2 Analysis

Our analysis is strongly inspired by the Monte Carlo interpretation of the PageRank random walk, even though our algorithms, explained in Section 4, do not depend on this interpretation. Hence, first we give some background about this interpretation. Following [3] one can write the PageRank (row) vector $\bar{\pi}^t$ of G^t as

$$\bar{\pi}^t = \frac{\epsilon}{n}\bar{1}^T \left(I - (1-\epsilon)P(t)\right)^{-1} = \frac{\epsilon}{n}\bar{1}^T \sum_{k\geq 0}(1-\epsilon)^k P(t)^k. \quad (1)$$

To have a random walk interpretation of this formula, as in [5], assume that we start one random walk from each node of G^t. At each step, each walk (independently) terminates with probability ϵ and otherwise follows the transition probabilities in $P(t)$. The sum of the expected lengths of the n walks is n/ϵ. Thus, an interpretation of the right hand of (1) is that π^t_u, the PageRank of node u, equals the expected number of visits of all the n walks to node u divided by the expected sum of lengths of the n walks.

Following this interpretation, for an edge (u, v), let $f^t_{u,v} = (1-\epsilon)\pi^t(u)P(t)_{u,v}$ be the amount of PageRank *flow* through the edge at time t. Then, similar to the relation between the PageRank of a node and the number of random walks visiting a node, $\frac{n}{\epsilon}f^t_{u,v}$ is the expected number of random walks traversing edge (u, v) in that process.

Using this interpretation, we prove our first lemma. We recall that the variation distance is half the L_1 distance.

LEMMA 1. *Let $D(\pi^{t+1}, \pi^t)$ be the total variation distance between π^{t+1} and π^t. Then,*

$$E[D(\pi^{t+1}, \pi^t)] \leq \frac{1-\epsilon}{m\epsilon}.$$

PROOF. Assume that G^{t+1} was obtained from G^t by changing edge (u, v) to edge (u, v'). Only walks that traversed edge (u, v) in G^t are affected by this change. Each such walk has expected length $1/\epsilon$ beyond edge (u, v) (or (u, v')) and the walk removed from one sets of nodes and added to another set of nodes. Thus, the sum of changes of the PageRank values of all the nodes from that change is bounded by

$$2f^t_{u,v}\frac{1}{\epsilon} = 2\pi^t_u(1-\epsilon)\frac{1}{\text{out}^t_u}\frac{1}{\epsilon}.$$

Now note that the edge to be removed at time t is chosen uniformly at random, and hence, for any node u, the probability of choosing an outgoing edge from u is out^t_u/m. Summing over all nodes, the expected change is then bounded

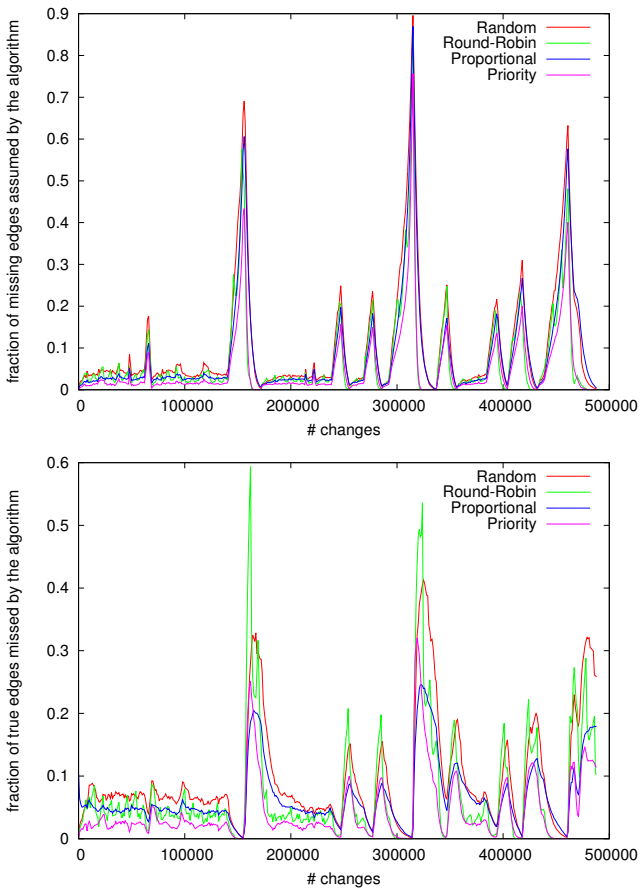

Figure 5: Staleness of the image of algorithms for the AS graph.

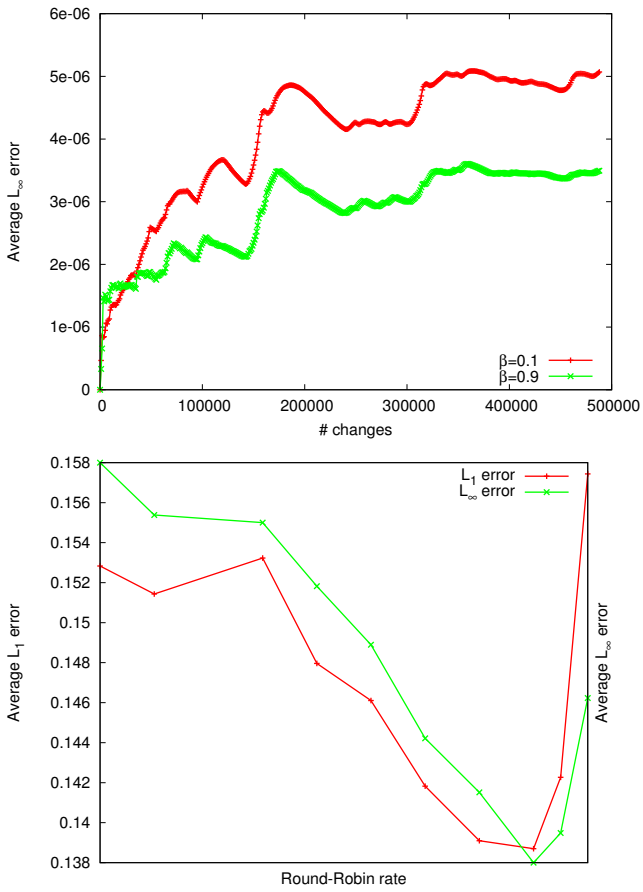

Figure 6: Average L_1 and L_∞ error of the hybrid algorithm as a function of the fraction β of round-robin probes for the AS graph.

by

$$2\sum_u \frac{\text{out}_u^t}{m}\pi_u^t(1-\epsilon)\frac{1}{\text{out}_u^t}\frac{1}{\epsilon} \le \frac{2(1-\epsilon)}{m\epsilon},$$

and the total variation distance is by definition half of this quantity. □

The above lemma shows that by probing the nodes in a round-robin fashion we maintain the total variation distance bounded by $\frac{n(1-\epsilon)}{m\epsilon}$. However, the variation distance argument does not give a better bound for the error in the PageRank of individual nodes. So, next we focus on better estimates of the change in the PageRank of individual nodes.

LEMMA 2. *The expected PageRank of any node x at time $t+1$, conditioned on the graph at time t, satisfies*

$$\pi_x^t\left(1-\frac{1}{\epsilon^2 m}\right) \le E[\pi_x^{t+1} \mid G^t] \le \pi_x^t\left(1+\frac{1}{\epsilon^2 m}\right).$$

PROOF. Similarly to the proof of the previous lemma, if G^{t+1} is obtained from G^t by changing edge (u,v) to edge (u,v'), only walks that traversed edge (u,v) are affected by this change, and each such walk has expected length $1/\epsilon$ beyond edge (u,v) or (u,v').

For a node x, we first bound the expected change in the PageRank of x from adding an edge pointing to it (i.e., the case $v'=x$). By our model definition, the probability of the new edge pointing to x is π_x^t, and the flow on that edge is the flow of an edge chosen uniformly at random from all m edges, i.e.,

$$\sum_u (1-\epsilon)\frac{\text{out}_u^t}{m}\pi_u^t\frac{1}{\text{out}_u^t} = \frac{1-\epsilon}{m}.$$

The expected number of times that the flow through the new edge can return to x is at most equal to the expected length of each walk passing through x, i.e., $1/\epsilon$, thus, the expected change in the PageRank of x from adding an edge pointing to it is bounded by $(1-\epsilon)\pi_x^t\frac{1}{m\epsilon}$.

Then, we bound the expected change in the PageRank of x from removing an edge pointing it (i.e., the case $v=x$). The probability of removing an edge pointing to x is in_x^t/m. Since this edge is picked uniformly at random, the flow on it is $\pi_x^t\frac{1}{\text{in}_x^t}$, and again the expected number of times that the flow through this edge could return to x is bounded by $1/\epsilon$. Thus the expected change in the PageRank of x from deleting an edge pointing to it at step t is bounded by $\pi_x^t\frac{1}{m\epsilon}$.

Hence, from the above two cases, the expected change in the PageRank of a node x from modifying (adding or deleting) an edge directly pointing to x at time t is bounded by $\pi_x^t\frac{1}{m\epsilon}$.

Next, we analyze the effect of modifying an arbitrary edge

on the PageRank of x. To do so, we note that since the PageRank vector satisfies the relation

$$\pi^t = \frac{\epsilon}{n}\bar{1}^T + (1-\epsilon)\pi^t P(t),$$

we can write for any $k \geq 1$,

$$\pi_x^t = \frac{\epsilon}{n}\sum_{j=0}^{k-1}(1-\epsilon)^j \sum_w P(t)_{w,x}^j + (1-\epsilon)^k \sum_w \pi_w^t P(t)_{w,x}^k, \quad (2)$$

where $P(t)_{w,x}^j$ is the (w,x)th entry in the jth power of the matrix $P(t)$; thus $P(t)_{w,x}^j > 0$ if and only if there is a path of length j from w to x in G^t. We know that if an edge pointing to a node w is modified, it changes π_w^t by at most $\pi_w^t \frac{1}{m\epsilon}$. So, if w is at distance k from u, we see from Equation (2) that this change affects π_x^t by at most $(1-\epsilon)^k \pi_w^t \frac{1}{m\epsilon} P(t)_{w,x}^k$. Thus, the expected change to the PageRank of x from adding or deleting an edge pointing to a node at distance k from it is bounded by $\pi_x^t(1-\epsilon)^k \frac{1}{m\epsilon}$. Summing over all distances bounds the change in PageRank of x by:

$$\sum_{k\geq 0} \pi_x^t (1-\epsilon)^k \frac{1}{m\epsilon} = \pi_x^t \frac{1}{\epsilon^2 m},$$

which finishes the proof. □

We remark that the above bound is tight only when x has exactly one outgoing edge that is a self-loop. A simple induction, using Lemma 2, then shows:

COROLLARY 3. *For any node x, time t, and time difference $\tau > 0$:*

$$\left(1 - \frac{1}{\epsilon^2 m}\right)^\tau \pi_x^t \leq E[\pi_x^{t+\tau} \mid G^t] \leq \left(1 + \frac{1}{\epsilon^2 m}\right)^\tau \pi_x^t.$$

Next, recalling our notation that H^t is the state of the algorithm's graph at time t and ϕ^t is its PageRank (row) vector, we proceed to our main result.

THEOREM 4. *For a time instance t, assume that there exists an $\alpha > 0$ such that for all nodes $v \in V$ and all $t-2n \leq \tau \leq t-1$:*

$$(1-\alpha)\phi_v^\tau \leq E[\pi_v^\tau \mid G^{\tau-1}, H^\tau] \leq (1+\alpha)\phi_v^\tau.$$

Then, letting $\beta = (1-\epsilon)\frac{1+\alpha}{m}(1+\frac{1}{\epsilon^2 m})^{2n}$, we have for all $v \in V$:

$$(1-\beta)\phi_v^t \leq E[\pi_v^t \mid G^{t-1}, H^t] \leq (1+\beta)\phi_v^t.$$

PROOF. Since the hybrid algorithm probes each node at least once in each interval of $2n$ steps, any edge in $H^t \setminus G^t$ or in $G^t \setminus H^t$ corresponds to a change in the last $2n$ steps before time t. For each node w let $\tilde{\phi}_w^t = \min_{i=t-2n}^{t-1} \phi_w^i$ denote the minimum value of ϕ_w in the $2n$ steps up to time t.

The head of an outgoing edge from v is changed in each step with probability $1/m$. The probability that the head of edge (v,u) was changed after the last probe of v is bounded by

$$\sum_{k\geq 0} \frac{1}{m}(1-\tilde{\phi}_v^t)^k = \frac{1}{\tilde{\phi}_v^t m}. \quad (3)$$

Let τ^* be the step in which $\phi_v^{\tau^*} = \tilde{\phi}_v^t$. By the theorem's assumption, we have

$$E[\pi_v^{\tau^*} \mid G^{\tau^*-1}, H^{\tau^*}] \leq (1+\alpha)\phi_v^{\tau^*},$$

and by Corollary 3,

$$E[\pi_v^t \mid G^{\tau^*}] \leq (1+\frac{1}{\epsilon^2 m})^{t-\tau^*}\pi_v^{\tau^*} \leq \left(1+\frac{1}{\epsilon^2 m}\right)^{2n} \pi_v^{\tau^*}.$$

Thus,

$$E[\pi_v^t \mid G^{\tau^*}, H^{\tau^*}] \leq (1+\alpha)\left(1+\frac{1}{\epsilon^2 m}\right)^{2n} \tilde{\phi}_v^t. \quad (4)$$

Hence, using Equation (3), we can bound the expected flow out of v not known to the algorithm by

$$E\left[\sum_x \frac{1}{\tilde{\phi}_v^t m} f_{v,x}^t\right] = E\left[\sum_x \frac{1}{\tilde{\phi}_v^t m}(1-\epsilon)\pi_v \frac{1}{\text{out}_v^t}\right] \leq \beta. \quad (5)$$

Applying the random walk interpretation of (1), we can obtain an upper bound for $E[\pi^t \mid G^{t-1}, H^t]$ as follows:

- We start one random walk from each node in H^t.
- For each random walk that traverses an edge in $H^t \setminus G^t$, we remove the part of the walk from that edge to the end of the random walk. Note that the remaining flow is counted both in π^t and ϕ^t.
- Let $\bar{\phi}_v^t$ be the flow through node v after the removal of parts of the random walks, then $E[\pi_v^t \mid G^{t-1}, H^t] \geq E[\bar{\phi}_v^t]$.

By (5) the expected flow removed from H^t can be estimated by applying (1) with a fraction β of a random walk starting at each node. Thus,

$$E[\bar{\phi}_v^t] = (1-\beta)\phi_v^t \leq E[\pi_v^t \mid G^{t-1}, H^t].$$

The other side of the inequality can be proved by applying the same walk fraction removal argument to G^t (instead of H^t). □

This gives us the final result in our analysis.

COROLLARY 5. *In the steady state*

$$\left(1 - O\left(\frac{1}{m}\right)\right)\phi^t \leq E[\pi^t \mid G^{t-1}, H^t] \leq \left(1 + O\left(\frac{1}{m}\right)\right)\phi^t.$$

PROOF. Follows from the observation that for α and β defined in Theorem 4, if $\alpha = \Omega(\frac{1}{m})$, then $\beta < \alpha$. □

This shows that assuming the evolving graph model, the approximate PageRank values returned by our algorithm are very close approximations of the true PageRank values, which finishes our analysis.

6.3 Extensions and variants

Even though we stated and proved our theoretical result in a simplified model, it can be generalized to much more general model using the same set of ideas. First, we note that the assumption that the number of edges of the graph does not change is not necessary, and our proof can be adapted to a model where edges addition and edge removal are two independent processes that add or remove edges at the same rate[§]. The proof is more technical, but the idea is the same, except now when an edge is removed, the paths that were

[§]It is necessary to assume that additions and removals occur at the same rate, since otherwise we will converge to an empty or a complete graph.

using that edge will be distributed among other outgoing edges of the node (instead of being rerouted to the edge that was just added).

Next, we note that similar results can be proved for a model of graphs with a growing node set. Here we need to be careful about the definition of the model. We assume that at each step, with probability $1 - \delta$ an edge is changed according to our edge change model and with probability δ, a new node is added. The new node will have outgoing links but no incoming links. In subsequent steps, the edge change model can add an incoming link to the new node. This is similar to the way networks like the web graph grow: sometimes a new web page is created; the page can contain links to other pages, but only over time it can attract links from other pages. The algorithm does not learn about the new page until it crawls a page that has an incoming link to it. Our theoretical result extends to this model as long as δ is small (hence there are not too many new nodes in the graph at any point), and the expected number of outgoing links from a new page is not large (otherwise the new pages can significantly affect the PageRank of other pages). We will leave the technical details to the full version of the paper.

Finally, we note that even though we proved the analysis for the algorithm that is a hybrid between Proportional Probing and Round Robin, with a little more work (and changes to the constants in the bound) it applies to the pure Proportional Probing method. The reason is that the PageRank of any node in the graph is always at least ϵ/n. Therefore, even without running Round Robin in alternate steps, in expectation each node is guaranteed to have been probed in the last n/ϵ steps.

7. CONCLUSIONS

In this paper we studied the problem of efficiently computing PageRank on an evolving graph. We obtained simple yet effective algorithms and demonstrated that they work well in practice on both real-world and synthetic graphs. We also analyzed the theoretical error bounds of the algorithm, assuming a particular model of graph evolution. While one can complain that the evolution model is specialized, our goal was to showcase the possibility of achieving the best of two worlds: a simple algorithm for a real-world problem and a provable guarantee under a model of the world using a non-trivial analysis. It remains a challenging problem to extend our theoretical analysis to other models of graph evolution. Studying other PageRank-like measures on evolving graphs is an interesting research direction.

8. REFERENCES

[1] A. Anagnostopoulos, R. Kumar, M. Mahdian, and E. Upfal. Sort me if you can: How to sort dynamic data. In *ICALP*, pages 339–350, 2009.

[2] A. Anagnostopoulos, R. Kumar, M. Mahdian, and E. Upfal. Algorithms on evolving graphs. In *ITCS*, pages 149–160, 2012.

[3] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte Carlo methods in Pagerank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.*, 45(2):890–904, 2007.

[4] B. Bahmani, K. Chakrabarti, and D. Xin. Fast personalized PageRank on MapReduce. In *SIGMOD*, pages 973–984, 2011.

[5] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized PageRank. *PVLDB*, 4:173–184, 2010.

[6] P. Berkhin. Survey: A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[7] Y.-Y. Chen, Q. Gan, and T. Suel. Local methods for estimating PageRank values. In *CIKM*, pages 381–389, 2004.

[8] S. Chien, C. Dwork, S. Kumar, D. Simon, and D. Sivakumar. Link evolution: Analysis and algorithms. *Internet Mathematics*, 1(3):277–304, 2003.

[9] A. Das Sarma, S. Gollapudi, and R. Panigrahy. Estimating PageRank on graph streams. In *PODS*, pages 69–78, 2008.

[10] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *WWW*, pages 421–430, 2007.

[11] J. V. Davis and I. S. Dhillon. Estimating the global PageRank of web communities. In *KDD*, pages 116–125, 2006.

[12] D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah, editor, *Algorithms and Theoretical Computing Handbook*. CRC Press, 1999.

[13] S. Kamvar, T. Haveliwala, and G. Golub. Adaptive methods for the computation of PageRank. Technical report, Stanford University, 2003.

[14] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Exploiting the block structure of the web for computing PageRank. Technical report, Stanford University, 2003.

[15] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation methods for accelerating PageRank computations. In *WWW*, pages 261–270, 2003.

[16] R. Kleinberg. *Online Decision Problems with Large Strategy Sets*. PhD thesis, MIT, 2005.

[17] A. N. Langville and C. D. Meyer. Updating PageRank using the group inverse and stochastic complementation. Technical report, NCSU, Mathematics Department, 2002.

[18] A. N. Langville and C. D. Meyer. Updating PageRank with iterative aggregation. In *WWW (posters)*, pages 392–393, 2004.

[19] A. N. Langville and C. D. Meyer. Updating Markov Chains with an eye on Google's PageRank. *SIAM J. Matrix Anal. Appl.*, 27(4):968–987, 2006.

[20] F. McSherry. A uniform approach to accelerated PageRank computation. In *WWW*, pages 575–582, 2005.

[21] S. Muthukrishnan. *Data Streams: Algorithms and Applications*. Now Publishers Inc, 2005.

[22] C. Olston and S. Pandey. Recrawl scheduling based on information longevity. In *WWW*, pages 437–446, 2008.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[24] S. Pandey and C. Olston. User-centric web crawling. In *WWW*, pages 401–411, 2005.

[25] S. Pandey and C. Olston. Crawl ordering by search impact. In *WSDM*, pages 3–14, 2008.

[26] D. Ron. Property testing: A learning theory perspective. *Found. Trends Mach. Learn.*, 1:307–402, 2008.

[27] D. Ron. Algorithmic and analysis techniques in property testing. *Found. Trends Theor. Comput. Sci.*, 5:73–205, 2010.

[28] H. H. Song, T. W. Cho, V. Dave, Y. Zhang, and L. Qiu. Scalable proximity estimation and link prediction in online social networks. In *IMC*, pages 322–335, 2009.

[29] H. Tong, S. Papadimitriou, P. S. Yu, and C. Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *SDM*, pages 704–715, 2008.

[30] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *WWW*, pages 136–147, 2002.